# למידת מכונה – תרגיל 2

**צחי אלחדד, 206214165**

1. **חלק מימושי**

- **אלגוריתם KNN:**

את אלגוריתם זה מימשתי לפי הגדרתו היחסית פשוטה, עברתי על כל דוגמא x' שנמצאת ב-test וחישבתי את k הדוגמאות הכי קרובות לה מתוך הדוגמאות ב-train, כמו כן על מנת למצוא את הכי קרובות השתמשתי בפונקציית מרחק אוקלידי (שמצאתי שנותנת לי תוצאות טובות וגבוהות לעומת מרחק מנהטן לדוגמא), וכן השתמשתי במיון על מערך המרחקים על מנת למצוא את k הנקודות הכי קרובות.
כמו כן, לאלגוריתם זה לא קיים תהליך של אימון.

- **אלגוריתם Preceptrom:**

לאלגוריתם זה יש שלב של אימון ושלב של חיזוי. כמו כן, יצרתי מטריצה של וקטורי משקלים W כמספר המחלקות שצריך לסווג (3 מחלקות – 3 וקטורי W).
**בשלב האימון** ביצעתי למידת של וקטורי המשקלים ע"י כך שעבור כל דוגמא ב-train הסתכלתי על התיוג שלה y, ביצעתי מכפלה של מטריצת המשקלים W בדוגמא ובחרתי בוקטור המשקל המקסימלי המייצג את המחלקה המנובאת. אם צדקתי בחיזוי ואכן המחלקה המנובאת הינה כמו y הנתון, לא ביצעתי עידכונים, **ואחרת**, ביצעתי עדכונים לוקטורי המשקלים כפי שנרשם לנו בדף התרגיל עבור כל אלגוריתם ובפרט לזה.

$$w_{t+1}^{y} = w_t^{y} + \eta * x$$

$$w_{t+1}^{\hat{y}} = w_t^{\hat{y}} - \eta * x$$

$$w_{t+1}^{i \neq \hat{y}, y} = w_{t+1}^{i \neq \hat{y}, y}$$

**בשלב החיזוי** עברתי על כל דוגמא ב-test שקיבלתי ובעזרת וקטורי המשקלים שלמדו בשלב האימון ניבאתי את התיוג עבור כל דוגמא (ע"י לקיחת אינדקס של המחלקה שהביאה את המקסימום במכפלה של וקטורי המשקלים עם הדוגמא).

- **<u>אלגוריתם SVM:</u>**

$$w_{t+1}^{y} = (1-\eta\lambda)w_t^{y} + \eta * x$$

$$w_{t+1}^{\hat{y}} = (1-\eta\lambda)w_t^{\hat{y}} - \eta * x$$

$$w_{t+1}^{i\neq\hat{y},y} = (1-\eta\lambda)w_{t+1}^{i\neq\hat{y},y}$$

המימוש דומה לאלגוריתם Preceptrom פרט לשלב האימון ובו **שלב עדכון וקטורי המשקלים של כל מחלקה.** באלגוריתם זה יש שימוש בפונקציית hinge loss כאשר אנו בודקים אם ה-loss גדול מ-0 ואם אכן כן סימן שטעינו בחיזוי ומעדכנים את וקטורי המשקלים. בנוסף מעדכנים גם את הוקטורי משקלים שלא היו קשורים לחיזוי (כלומר לא רק הוקטור של y ו-y').

- **<u>אלגוריתם PA:</u>**

$$\tau = \frac{\ell(w,x,y)}{2\cdot||x||^2}$$

המימוש דומה לאלגוריתם Preceptrom פרט לשלב האימון ובו **שלב עדכון וקטורי המשקלים של כל מחלקה.** גם באלגוריתם זה יש שימוש בפונקציית loss אך השתמשתי בו על מה לעדכן את ההיפר פרמטר טאו.

- **<u>נרמול ערכים:</u>**

בניתי שתי פונקציות נרמול- Minmax, ZScore:

```python
#   MinMax normalization function
def minMaxNormalization(list_train_x):
    num_features = len(list_train_x[0])
    new_train_x = np.zeros((len(list_train_x), num_features))
    for i in range(num_features):
        column = list_train_x[:, i]
        x_max = max(column)
        x_min = min(column)
        new_train_x[:, i] = (list_train_x[:, i] - x_min) / (x_max - x_min)
    return new_train_x
```

```python
#   ZScore normalization function
def zScoreNormalization(list_train_x):
    num_features = len(list_train_x[0])
    new_train_x = np.zeros((len(list_train_x), num_features))
    for i in range(num_features):
        column = list_train_x[:, i]
        x_mean = np.mean(column)
        x_std = np.std(column)
        new_train_x[:, i] = (list_train_x[:, i] - x_mean) / x_std
    return new_train_x
```

עבור אלגוריתם KNN מצאתי כי נרמול מוריד לי את אחוזי ההצלחה ולכן לא השתמשתי בנרמול.
עבור שאר האלגוריתמים גיליתי כי נרמול ZScore מעלה לי את אחוזי ההצלחה בצורה אופטימלית ולכן השתמשתי בנרמול זה.

- **Bias:**
השתמשתי ב-bias עבור הדוגמאות שנמצאות ב-train ו-test, הוספתי עמודה חדשה ובה מאותחלים הערכים ל-1, וכמו כן גם לוקטורי המשקלים w.
- **בחירת epochs:**
בחרתי מספר ephocs בכל אלגוריתם ע"י ניסוי וטעיה, בדקתי ממספר קטן והתקדמתי עד הגעה למספר epochs שבו האלגוריתם הינו יציב מבחינת הצלחות.
- **בדיקות:**
בניתי פונקציית CrossValidation על מנת לבדוק את טיב האלגוריתמים שבניתי (KNN,PA...), כלומר ביצוע ה-validation שלי היה חלוקת האימון ל-k חלקים (5/10) וכל פעם k-1 החלקים שימשו כמודל עבור האימון והחלק האחרון שימש ב-test. כמו כן גם ביצעתי חלוקה של 70%,30% כאשר 70 הינו חלק האימון ו-30 חלק ה-test, כלומר ביצעתי את תהליך ה-validation. וכן גם ביצעתי ערבוב shuffle על הדוגמאות כדי לבדוק שהערכים שלי באלגוריתמים לא תלויים בסדר הדוגמאות שהתקבלו ב-train.

```
#   Cross Validation/KFolds function
def crossValidation(train_x_list, train_y_list, k=5, test_index=0):
    #   Number of examples
    size = len(train_x_list)
    test_split_x = []
    test_split_y = []
    train_split_x = list()
    train_split_y = list()
    data_x = np.array_split(train_x_list,k)
    data_y = np.array_split(train_y_list,k)
    for i in range(k):
        if i == test_index:
            test_split_x = data_x[test_index]
            test_split_y = data_y[test_index]
            continue
        for vec in data_x[i]:
            train_split_x.append(vec)
        for label in data_y[i]:
            train_split_y.append(label)
```

כמו כן, לכל מחלקה של אלגוריתם בניתי פונקציה של accuracy על מנת למצוא אחוזי הצלחה בחיזוי דוגמאות, וכך יכולתי לבדוק את אחוזי הצלחת האלגוריתמים בהתאם לדוגמאות ה-test:

```
def accuracy(self, test_y):
    counter = 0
    prediction = self.predict()
    index = 0
    for y in test_y:
        if y != prediction[index]:
            counter += 1
        index += 1
    return 1 - float(counter / len(test_y))
```

## 2. היפרפרמטרים

- **עבור KNN:**

מציאת ה-k המתאים עבור האלגוריתם התבצעה ע"י פונקציה שבניתי שמוצאת את אחוז ההצלחות הגבוה ביותר מבין כל ה-kים שרצים מ-1 ועד שורש מספר הדוגמאות ומחזירה את ה-k האופטימלי (בtrain זה גיליתי כי ה-k האופטימלי הוא 9).
כמו כן גם לאחר חקירת הנושא גיליתי כי נהוג להציב ב- k את הערך- שורש מספר הדוגמאות וזה באמת הביא לאחוזי הצלחה גבוהים.

```
#   Returns the optimal k in KNN algorithm
def get_k_optimal():
    n = len(list_examples_x)
    k = np.sqrt(n)
    k = round(k + (0.2 * n))
    index_separate = round(n * 0.7)
    train_x_ls = list_examples_x[:index_separate]
    test_ls = list_examples_x[index_separate:]
    train_y_ls = list_examples_y[:index_separate]
    real_y = list_examples_y[index_separate:]
    max_val = 0
    max_k = 0
    for i in range(k):
        i += 1
        count = 0
        _knn = KNN(train_x_ls, train_y_ls, test_ls, i)
        ls_prediction = _knn.predict()
        for j in range(len(ls_prediction)):
            if ls_prediction[j] == real_y[j]:
                count += 1
        t_p_percent = count / len(ls_prediction)
        if max_val < t_p_percent:
            max_val = t_p_percent
            max_k = i

    return max_k
```

- **<u>עבור Preceptrom:</u>**
עבור אלגוריתם זה קבעתי את ערך ה-learning rate כך שבכל איטרציה(epoch) הוא מעודכן באופן הבא: (מס' איטרציה = i)

```
self.lr = 1 / np.sqrt(i + 1)
```

קבעתי זאת לפי דבריו של יוסי מההרצאה(המלצה של יוסי) ואכן זה הביא לאחוזי תוצאות גבוהים, וכן גם בדקתי ערכים נוספים.
בשיטה זאת האלגוריתם הצליח ללמוד את מידע האימון בצורה מדויקת והדרגתית.

- **<u>עבור SVM:</u>**
עבור אלגוריתם זה קיימים שני היפרפרמטרים-
  - Learning rate – lr
  - Lambda

**את ערך lr** קבעתי להיות 0.01 מכיוון שלאחר מס' בדיקות הביא לאחוזי תוצאות גבוהים במיוחד.
**את ערך lambda** קבעתי להיות מעודכן בכל איטרציה באופן הבא: (מס' איטרציה = i)

```
self._lambda = 1 / (i + 1)
if self._lambda < 0.0001:
    self._lambda = 0.0001
```

לאחר מס' בדיקות פונקציה זאת הביאה לאחוזי תוצאות גבוהים וכן שמתי לב שיש לעצור את ערכו של lambda מלקטון מיותר מ-0.0001 שכן הוא כבר מאבד מהאפקטיביות שלו.

- **<u>עבור PA:</u>**
עבור אלגוריתם זה קיימים שני היפרפרמטרים-
  - Learning rate – lr שהוספתי בגלל ה-epochs
  - Taho

**את ערך ה-lr** קבעתי להיות 0.1 בהתחלה ובכל עשירית מהאיטרציות עידכנתי אותו באופן הבא:

```
if i == temp_update_eta:
    temp_update_eta = temp_update_eta + self.epochs / 10
    self.lr = self.lr / 2
```

בכל עשירית מהאיטרציות חילקתי את lr ב-2 וגרמתי ללמידה להיות הדרגתית יותר ומדויקת וזה הביא לאחוזי הצלחה גבוהים במיוחד.

**את ערך טאו** קבעתי להיות באופן הבא לפי חישובים מתמטיים שנראו במהלך התרגול וההרצאה:

```
self.taho = loss / (np.power(np.linalg.norm(x), 2) * 2)
```

כמו כן הפונקציית loss הינה מוגדרת כפי שנלמד בהרצאה ובתרגול:

```
def loss_func(self, x, w_y, w_y_max):
    return max(0, 1 - np.dot(w_y, x) + np.dot(w_y_max, x))
```